IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–25. DOI:XXXXXXXX

DAWN′: Improved DAWN and Accelerated Implementation on ARM Cortex-M4

Junhao Huang[1*], Yijian Liu[2,3*], Yu Zhang[2,3], Xianhui Lu[2,3], Donglong Chen[4], Haiyang Xue[1†]

[1] Singapore Management University, Singapore jhhuang_nuaa@126.com,haiyangxue@smu.edu.sg
[2] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
liuyijian@iie.ac.cn,zhangyu1999@iie.ac.cn,luxianhui@iie.ac.cn
[4] Guangdong Provincial Key Laboratory IRADS, Beijing Normal-Hong Kong Baptist University, Zhuhai, China donglongchen@bnbu.edu.cn

Abstract. DAWN is a compact and efficient NTRU-based key encapsulation mechanism (KEM) presented at ASIACRYPT 2025. Its compact ciphertexts, below 512 bytes, make it particularly attractive for resource-constrained Internet-of-Things (IoT) deployments with stringent communication, memory, and computational constraints. In this paper, we revisit DAWN from both algorithmic and implementation perspectives and propose DAWN′, a refined variant with improved robustness and efficiency.

On the algorithmic side, we revise the decryption procedure and derive a rigorous upper bound on its decryption-failure rate (DFR), providing stronger correctness guarantees than the original construction. We further introduce tailored sparse ternary distributions that can be sampled using only simple bitwise operations, reducing the cost of polynomial sampling. On the implementation side, we present a comprehensive evaluation of DAWN′ on the ARM Cortex-M4. We optimize arithmetic over R_q using improved Plantard arithmetic and extensive lazy reduction, targeting the high-degree base multiplication and inversion induced by incomplete NTT. For high-degree base multiplication, we further reduce the number of multiplications by adapting a register-reshuffling strategy. We also adapt the radix-16 representation to accelerate binary polynomial multiplication over R_2, which arises from DAWN's zero-divisor encoding, and design dedicated radix-16 routines for the fast inversion used during key generation.

By combining these algorithmic and implementation techniques, our DAWN′ KEM implementation on Cortex-M4 achieves speedups of 1.78–2.54× and 1.94–3.96× over the reference implementation of DAWN′ and the original DAWN implementation, respectively. Comparisons with state-of-the-art MLWE- and NTRU-based KEM implementations demonstrate that DAWN′ offers a favorable combination of compact keys and ciphertexts, strong security guarantees, and high computational efficiency, making it a promising candidate for resource-constrained devices.

Keywords: Lattice-based Cryptography · KEM · NTRU · DAWN · IoT

1 Introduction

With the formal standardization of primary post-quantum cryptographic (PQC) primitives by NIST (such as ML-KEM and ML-DSA) [Nat24a, Nat24b] and the subsequent mandates

*These authors contributed equally to this work.
†Corresponding author.

Licensed under Creative Commons License CC-BY 4.0.

for cryptographic transition, the focus of PQC research has rapidly shifted from theoretical security to the practicalities of widespread migration. A critical frontier in this migration is the Internet-of-Things (IoT) ecosystem, where billions of legacy and resource-constrained embedded devices must be upgraded to withstand quantum adversaries. Unlike enterprise servers or desktop environments, IoT nodes operate under severe hardware boundaries, such as limited memory, power budgets, and tightly constrained network stacks.

In IoT devices, the significantly large public-key and ciphertext sizes inherent in lattice-based cryptography are a primary bottleneck. Achieving optimal ciphertext compactness has become a critical deployment constraint for key-encapsulation mechanisms (KEMs) on constrained IoT platforms. Bluetooth Low Energy (BLE) provides a representative example: the Attribute Protocol (ATT) limits an attribute value to 512 octets, while values exceeding the negotiated ATT maximum transmission unit must be transferred using multiple protocol data units (PDUs) [Blu23]. Thus, a ciphertext below 512B can, in principle, be represented by a single ATT attribute value, even though it may still be fragmented by lower protocol layers. This object-size target is particularly relevant when public keys are provisioned or cached, whereas a fresh ciphertext is transmitted for each key establishment.

However, many lattice-based KEMs exceed the 512B target even at their smallest parameter sets. For example, ML-KEM-512 has a 768B ciphertext [Nat24a], and the category-1 parameter sets of NTRU-HPS/HRSS also have ciphertexts larger than 512B [HRSS17]. Such ciphertexts may require application-level segmentation, additional reassembly state, or a transport abstraction capable of handling larger cryptographic objects.

Compact KEMs with ciphertexts below 512B are therefore an attractive design point for constrained deployments. Two prominent compact NTRU-based KEMs are BAT [FKPY22] and DAWN [LZL+25], whose category-1 ciphertexts have sizes of 473B and 436B, respectively. BAT achieves compactness through an NTRU trapdoor and a dedicated decoder that permits the use of a small modulus. Its encapsulation and decapsulation algorithms remain efficient, but the construction shifts substantial complexity to trapdoor key generation. While acceptable for scenarios using pre-provisioned static keys, the heavy key generation is the bottleneck when keys must be generated dynamically on the constrained devices.

This limitation makes the alternative design of DAWN [LZL+25] highly compelling for general IoT deployments. Without relying on a heavy trapdoor, DAWN obtains compact ciphertexts through zero-divisor encoding and a double-encoding framework. However, this design introduces a unique implementation consequence: in addition to modular polynomial arithmetic in R_q, its key generation, message decoding, and decryption procedures also require performance-critical binary polynomial arithmetic in R_2. This departs from previous NTRU-based schemes, which typically rely on arithmetic over ternary polynomial quotient rings, e.g., $R_3 = \mathbb{F}_3[x]/(x^p - x + 1)$ in NTRU Prime [BCLvV17]. DAWN therefore exposes two distinct arithmetic domains on its critical path. Modular arithmetic in R_q is well suited to incomplete number-theoretic transforms (NTTs), Plantard reduction, and the 16-bit DSP instructions of the ARM Cortex-M4 [AHKS22, HZZ+22, HAZ+24], whereas arithmetic in R_2 favors bit-packed representations and binary polynomial algorithms, as shown in many previous code-based schemes [CC21, CCK21, CGKT22, LJKH25]. Thus, optimizing both polynomial arithmetic in R_q and R_2 is essential for the practical deployment of DAWN′ on resource-constrained devices.

While DAWN is promising for IoT platforms, it introduces a correctness-analysis challenge in its design. Specifically, its original decryption-failure-rate (DFR) estimate is obtained from the coefficient-wise marginal distributions together with an independence approximation across the coefficients of polynomial products. Recent analyses have shown that convolution-induced dependencies can produce heavier tails in the rare-event regime, making an independence-based estimate optimistic in some parameter ranges [LGZ+26,

CLWL26]. Because the DFR contributes directly to the correctness and concrete CCA security, a conservative parameter set requires a rigorous bound that does not rely on this approximation.

Motivation and contributions. Addressing this theoretical gap of DAWN while preserving physical efficiency on constrained platforms is a central objective of this work. We address both aspects through DAWN′, a less heuristic and implementation-aware revision of DAWN targeting the ARM Cortex-M4. Our contributions can be summarized as follows.

- We revisit the DAWN scheme from both algorithmic and parameter perspectives and propose DAWN′. On the algorithmic side, we modify the decryption algorithm and provide a rigorous upper bound on the DFR, replacing the original heuristic DFR estimate relying on an independence assumption. The remaining assumption is the independence of the error introduced by compression, which is consistent with ML-KEM. On the parameter-selection side, we use specific sparse ternary distributions that can be sampled using only simple bitwise operations, thereby improving sampling efficiency and enabling constant-time implementation.
- We extend an incomplete-NTT multiplication strategy previously developed for smaller polynomial degrees to the higher-degree polynomial ring in DAWN′. For the resulting high-degree base multiplication and inversion, we develop an aggressive lazy-reduction schedule enabled by the comparatively small modulus and the adaptation of improved Plantard arithmetic. We further adapt a register reshuffling strategy, previously used in packed schoolbook multiplication, to the twisted length-k base multiplication induced by the incomplete NTT. We derive exact instruction counts to quantify the resulting savings of this method for base multiplication.
- We adopt the radix-16 representation for the polynomial arithmetic over R_2 for DAWN′ and explore different optimization strategies for polynomial multiplication with various small degrees. We then redesign the fast inversion over R_2 routines to operate directly on the radix-16 representation, avoiding repeated conversions between different polynomial representations.

By combining the above algorithmic and implementation optimizations, our DAWN′ implementation achieves speedups of 1.78–2.54× and 1.94–3.96× over the reference implementation of DAWN′ and the original DAWN implementation [LZL+25], respectively, on the ARM Cortex-M4. Our implementations are open-source in https://github.com/JunhaoHuang/dawnprime.

Relation to prior techniques. As DAWN′ has similar ring structures to other lattice-based schemes, a number of existing optimizations could apply here. We inherit the NTT/INTT layer-merging strategy and improved Plantard arithmetic modulo 769 in [HZZ+22, HAZ+24], lazy reduction for degree-2 base multiplication in [ABCG20], register reshuffling for packed schoolbook multiplication in [KMRV18, KRS19], the radix-16 representation for binary polynomials in [CC21, SKLE25], and the group decomposition of the DFR analysis in [PSH+25].

None of these transfers unchanged to the parameter regimes DAWN′ requires, and the adaptations are our contributions. We derive the range analysis that makes aggressive lazy reduction in high-degree base multiplication admissible for degree up to eight. We specialize register reshuffling from untwisted schoolbook multiplication to the twisted ring $Z_q[x]/(x^k - r)$, with exact instruction counts showing that the technique is break-even at $k = 4$ and profitable only for $k > 4$. We optimize high-degree base inversion with the Plantard-based lazy-reduction schedule. We present optimized small-degree radix-16 kernels together with a dedicated asymmetric $n/4 \times n/2$ routine for DAWN′. And we extend

the radix-16 representation beyond multiplication to the entire fast-inversion pipeline of DAWN′ in Subsubsection 4.3.2: exact division by x^m+1 becomes a strided prefix-exclusive-OR (eor) over 4-bit lanes, and radix conversion is fused into the post-sampling reduction, so the pipeline never leaves the packed representation.

As for the DFR analysis, we observe that the group decomposition in [PSH+25] aligns particularly well with the zero divisors used in DAWN. Accordingly, we incorporate this technique into DAWN′ and combine the correct-decoding condition with the failure probability of SimpleDecoding to obtain an efficient and straightforward DFR estimate.

2 Preliminaries

In this section, we briefly introduce the DAWN, its core operations, and ARM Cortex-M4 architecture.

2.1 Notations

We denote the polynomial by bold lower-case letters. For a polynomial $\boldsymbol{a} = \sum_{i=0}^{n-1} \boldsymbol{a}_{[i]} x^i$, the coefficient of x^i is written $\boldsymbol{a}_{[i]}$, and $\|\boldsymbol{a}\|_\infty = \max_{0\le i<n} |\boldsymbol{a}_{[i]}|$. We use the function Sample($\mathcal{D}$) to denote the sampling process of an element drawn from the distribution $\mathcal{D}$, and we use the notation Sample($\mathcal{D}, \rho$) to denote the sampling process with the given random seed ρ. $\mathcal{U}(S)$ denotes the uniform distribution over the finite set S.

2.2 DAWN Key Encapsulation Mechanism

DAWN [LZL+25] is a recently proposed NTRU-based key encapsulation mechanism (KEM) that is designed to be secure against quantum attacks. Its security relies on the hardness of the NTRU and RLWE problems. The core design is to use zero divisors to reduce amplification, and to employ two nested zero divisors to enable simple one-wrapped-error correction. In the default setting, the two zero divisors are chosen as $x^{n/2}+1$ and $x^{n/4}+1$ over the ring $\mathbb{F}_2[x]/(x^n+1)$. The auxiliary algorithms FastInversion and DAWN.SimpleDecoding are detailed in Algorithm 1–Algorithm 2.

Algorithm 1 FastInversion($\boldsymbol{f}, d$)

Input: Polynomial $\boldsymbol{f}$, degree d
Output: Inversion $\boldsymbol{f}_{\text{inv}}$ or $\perp$
1: **if** $\boldsymbol{f} \equiv 0 \pmod{\langle 2, x+1\rangle}$ **then**
2: **return** $\perp$
3: **end if**
4: $l \leftarrow \log_2(d)$
5: $\boldsymbol{k} \leftarrow (\boldsymbol{f}+1)/(x+1)$
6: $\boldsymbol{f}_{\text{inv}} \leftarrow 1$
7: **for** $i = 0$ to $l-1$ **do**
8: $\boldsymbol{b} \leftarrow \boldsymbol{k}\boldsymbol{f}_{\text{inv}} \pmod{\langle 2, x^{2^i}+1\rangle}$
9: $\boldsymbol{k} \leftarrow \boldsymbol{k} + \boldsymbol{f}\boldsymbol{b} \pmod{\langle 2, x^d+1\rangle}$
10: $\boldsymbol{k} \leftarrow \boldsymbol{k}/(x^{2^i}+1)$
11: $\boldsymbol{f}_{\text{inv}} \leftarrow \boldsymbol{f}_{\text{inv}} + (x^{2^i}+1)\boldsymbol{b}$
12: **end for**
13: **return** $\boldsymbol{f}_{\text{inv}}$

The DAWN.PKE algorithms are presented in Algorithm 3, Algorithm 4, and Algorithm 5. We rewrite the original algorithms to make the NTT procedure explicit,

Algorithm 2 DAWN.SimpleDecoding($\boldsymbol{a}, \boldsymbol{m}', \boldsymbol{f}_{\text{inv}}$)

Input: Three polynomials $\boldsymbol{a}, \boldsymbol{m}', \boldsymbol{f}_{\text{inv}}$
Output: Message polynomial $\boldsymbol{m}$
1: $\boldsymbol{e}' \leftarrow \boldsymbol{a} \pmod{\langle 2, x^{n/4}+1\rangle}$
2: $\boldsymbol{a}' \leftarrow ((\boldsymbol{e}')^{-1}\boldsymbol{a}) \pmod{\langle x^n+1\rangle}$
3: $i \leftarrow 0$
4: $base \leftarrow q$
5: **for** j **from** 0 **to** 3 **then**
6: $\quad v \leftarrow \frac{q}{2} - |\boldsymbol{a}'_{[jn/4]}|$
7: $\quad$ **if** $v < base$ **then**
8: $\quad\quad base \leftarrow v$
9: $\quad\quad i \leftarrow jn/4$
10: $\quad$ **end if**
11: **end for**
12: $\boldsymbol{m} \leftarrow \boldsymbol{m}' + x^i \boldsymbol{f}_{\text{inv}} \boldsymbol{e}' \pmod{\langle 2, x^{n/2}+1\rangle}$
13: **if** $\boldsymbol{e}' = 0$ **then**
14: $\quad \boldsymbol{m} \leftarrow \boldsymbol{m}' \pmod{\langle 2, x^{n/4}\rangle}$
15: **else**
16: $\quad \boldsymbol{m} \leftarrow \boldsymbol{m} \pmod{\langle 2, x^{n/4}\rangle}$
17: **end if**
18: return $\boldsymbol{m}$

and replace the encoding term $(x^{n/2}+1)^{-1}(x^{n/4}+1)$ with the equivalent expression $\frac{q+1}{2}(1+x^{n/4}+x^{n/2}+x^{3n/4})$ over the ring $\mathbb{F}_2[x]/(x^{n/2}+1)$. Steps 6 and 12 in Algorithm 3 are used to check invertibility in the NTT domain. The public parameter M_c in Algorithm 4 and Algorithm 5 is the coefficient width of the compressed ciphertext.

Algorithm 3 DAWN.PKE.KeyGen(ρ_g, ρ_f)

Input: Seeds ρ_g, ρ_f
Output: Public key pk, secret key sk_{PKE}
1: $N \leftarrow 0$
2: **repeat**
3: $\quad \boldsymbol{g} \leftarrow \text{Sample}(\mathcal{D}_g^n, \text{PRF}(\rho_g, N))$
4: $\quad \widehat{\boldsymbol{g}} \leftarrow \text{NTT}(\boldsymbol{g})$
5: $\quad N \leftarrow N+1$
6: **until** $\boldsymbol{g}$ is invertible in the quotient ring $\mathbb{Z}_q[x]/(x^n+1)$
7: **repeat**
8: $\quad \boldsymbol{f} \leftarrow \text{Sample}(\mathcal{D}_f^n, \text{PRF}(\rho_f, N))$
9: $\quad \boldsymbol{f}_{\text{inv}} \leftarrow \text{FastInversion}(\boldsymbol{f}, n/4)$
10: $\quad \widehat{\boldsymbol{f}} \leftarrow \text{NTT}(\boldsymbol{f})$
11: $\quad N \leftarrow N+1$
12: **until** $\boldsymbol{f}$ is invertible in the quotient ring $\mathbb{Z}_q[x]/(x^n+1)$ and $\boldsymbol{f}_{\text{inv}} \neq \perp$
13: $\widehat{\boldsymbol{h}} \leftarrow \widehat{\boldsymbol{g}} \odot \widehat{\boldsymbol{f}}^{-1} \pmod{\langle q, x^n+1\rangle}$
14: $pk \leftarrow \widehat{\boldsymbol{h}}$
15: $sk_{\text{PKE}} \leftarrow (\widehat{\boldsymbol{f}}, \boldsymbol{f}_{\text{inv}})$
16: **return** (pk, sk_{PKE})

We note that DAWN.SimpleDecoding in Algorithm 2, as well as the corresponding DFR analysis of DAWN, relies on assumptions about the coefficient distribution of certain polynomial products, which makes its DFR estimate heuristic.

Algorithm 4 DAWN.PKE.Encrypt$(pk, \boldsymbol{m}, \rho)$

Input: Public key pk, message $\boldsymbol{m}$, seed ρ
Output: Ciphertext ct

1: $\widehat{\boldsymbol{h}} \leftarrow pk$
2: $\boldsymbol{s} \leftarrow \text{Sample}(\mathcal{D}_s^n, \text{PRF}(\rho, 0))$
3: $\boldsymbol{e} \leftarrow \text{Sample}(\mathcal{D}_e^n, \text{PRF}(\rho, 1))$
4: $\widehat{\boldsymbol{s}} \leftarrow \text{NTT}(\boldsymbol{s})$
5: $\boldsymbol{u} \leftarrow \text{InverseNTT}(\widehat{\boldsymbol{h}} \odot \widehat{\boldsymbol{s}}) + \boldsymbol{e} \pmod{\langle q, x^n + 1\rangle}$
6: $\boldsymbol{c} \leftarrow \boldsymbol{u} + \frac{q+1}{2}(1 + x^{n/4} + x^{n/2} + x^{3n/4})\boldsymbol{m} \pmod{\langle q, x^n + 1\rangle}$
7: $ct \leftarrow \text{Compress}(\boldsymbol{c}, q, M_c)$
8: **return** ct

Algorithm 5 DAWN.PKE.Decrypt(sk_{PKE}, ct)

Input: Secret key sk_{PKE}, ciphertext ct
Output: Message $\boldsymbol{m}$

1: $(\widehat{\boldsymbol{f}}, \boldsymbol{f}_{\text{inv}}) \leftarrow sk_{\text{PKE}}$
2: $\boldsymbol{c}^\dagger \leftarrow \text{Decompress}(ct, q, M_c)$
3: $\widehat{\boldsymbol{c}} \leftarrow \text{NTT}(\boldsymbol{c}^\dagger)$
4: $\boldsymbol{a} \leftarrow (x^{n/2} + 1)\,\text{InverseNTT}(\widehat{\boldsymbol{c}} \odot \widehat{\boldsymbol{f}}) \pmod{\langle q, x^n + 1\rangle}$
5: $\boldsymbol{m}' \leftarrow \boldsymbol{a}\boldsymbol{f}_{\text{inv}} \pmod{\langle 2, x^{n/2} + 1\rangle}$
6: $\boldsymbol{m} \leftarrow \text{DAWN.SimpleDecoding}(\boldsymbol{a}, \boldsymbol{m}', \boldsymbol{f}_{\text{inv}})$
7: **return** $\boldsymbol{m}$

DAWN.KEM follows the same Fujisaki–Okamoto (FO) transformation with implicit rejection used in ML-KEM [Nat24a]. We refer readers to [LZL+25] for the detailed description of DAWN.KEM.

2.3 Core Operations in DAWN

DAWN [LZL+25] proposes two parameter settings, denoted by α and β. The α parameter set minimizes the ciphertext size and uses modulus $q = 769$, whereas the β parameter set minimizes the combined size of the ciphertext and public key and uses modulus $q = 257$. However, DAWN-β adopts less conservative parameters, leaving smaller margins in the concrete security analysis, and is therefore less suitable for standardization. Consequently, we focus on DAWN-α in this work.

A close examination of the DAWN.PKE algorithms shows that the dominant time-consuming operations are NTT-based polynomial multiplications over R_{769}, together with binary polynomial multiplications over R_2. The latter arise in FastInversion, in Step 5 of Algorithm 5, and in Step 12 of Algorithm 2. Apart from these arithmetic operations, the additional cost of sampling mainly comes from hash-function evaluations.

The compression used in DAWN is simple rounding, i.e., $\text{Compress}(c, q, M_c) = \lfloor c \cdot M_c/q \rceil$, and the corresponding $\text{Decompress}(ct, q, M_c) = \lfloor ct \cdot q/M_c \rceil$. We modify them as shown in Section 3.2.

Similar to other RLWE-based and NTRU-based constructions, the most expensive arithmetic operations in DAWN are polynomial multiplications and inversions in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ where $q = 769$ and n is a power of two. DAWN uses $n = 512$ and $n = 1024$ for its two security levels. Since $q - 1 = 768$ is divisible by 256, polynomial multiplication in R_q can be implemented using an incomplete 7-level NTT, followed by an inverse NTT (INTT), in a manner similar to Kyber. Specifically, the NTT modulo 769 performs seven layers of butterfly operations, transforming a polynomial of length 512 or 1024 into small base elements of length 4 or 8, respectively.

Let $k = n/128$. After the incomplete NTT, the base multiplication computes

$$\hat{c}(x) = \hat{a}(x)\hat{b}(x) \bmod (x^k - r),$$

where r is the corresponding twiddle factor. Since $x^k \equiv r \pmod{x^k - r}$, the coefficient $\hat{c}_{[i]}$ is given by

$$\hat{c}_{[i]} = \left(\sum_{j=0}^{i} \hat{a}_{[j]} \hat{b}_{[i-j]} + r \sum_{j=i+1}^{k-1} \hat{a}_{[j]} \hat{b}_{[i+k-j]} \right) \bmod q, \qquad 0 \le i < k. \tag{1}$$

Finally, the INTT is applied to $\mathbf{c}$ to obtain the product in R_q.

Moreover, due to the use of zero-divisor encoding and the double-encoding paradigm, DAWN does not rely solely on NTT-based arithmetic over R_q. It also requires a large number of polynomial multiplications in the binary ring

$$R_2 = \mathbb{F}_2[x]/(x^m + 1), \qquad m = 2, 4, 8, \ldots, \frac{n}{2}.$$

In the reference DAWN implementation, multiplication in R_2 is performed using schoolbook multiplication with eor-based accumulation. However, this approach becomes inefficient for larger m because of its $O(m^2)$ complexity. Furthermore, the fast inversion over R_2 and simple decoding procedures used in [LZL+25] rely heavily on binary polynomial multiplication. Therefore, optimizing polynomial multiplication over R_2 is crucial for improving the overall performance of DAWN.

2.4 ARM Cortex-M4

ARM Cortex-M4 is a 32-bit microcontroller processor based on the ARMv7E-M architecture, and is widely deployed in embedded systems and IoT devices. It provides a compact instruction set covering arithmetic, logical, bit-manipulation, shift/rotation, and load/store operations. On Cortex-M4, most integer arithmetic and bitwise instructions have single-cycle latency, whereas division is variable-latency and is not constant-time. It also provides a DSP extension, including specialized, extended single-cycle multiply-accumulate (MAC) instructions like umull, smuad{x}, smulw{b,t}, and smul{b,t}{b,t}, which will be widely used in this paper. In contrast, memory-access instructions are relatively expensive, as they may introduce additional latency and pipeline stalls. Therefore, reducing load/store operations is particularly important for efficient cryptographic implementations.

Another useful architectural feature is the inline barrel shifter in data-processing instructions. It allows one operand to be shifted or rotated as part of an arithmetic or logical instruction, enabling certain rotations to be merged into subsequent computations without issuing separate rotation instructions. Moreover, Cortex-M4 implementations equipped with the floating-point extension provide 32 single-precision 32-bit floating-point (FP) registers. They can be used as an auxiliary register file to cache intermediate integer values, thereby reducing stack usage and memory-access overhead.

3 DAWN′

In this section, we present DAWN′, a refined version of the DAWN key encapsulation mechanism (KEM) that incorporates several optimizations to enhance its security and performance.

3.1 Refinement of DAWN: DAWN′

DAWN′.PKE inherits DAWN.PKE, except that DAWN.SimpleDecoding is replaced by DAWN′.SimpleDecoding, which is detailed in the following section. Likewise, DAWN′.KEM inherits DAWN.KEM, and we omit the description here. Thus, the CPA/CCA security proof is the same as DAWN in [LZL+25].

3.1.1 Remove Independence Assumption of Polynomial Convolution

To clarify the modification to the decryption of DAWN′, we first introduce the heuristic used in DAWN and then analyze the decryption failure rate (DFR). Let

$$\boldsymbol{c}^{\dagger} = \boldsymbol{hs} + \boldsymbol{e} + \frac{q+1}{2}(1 + x^{n/4} + x^{n/2} + x^{3n/4})\boldsymbol{m} + \boldsymbol{e}_{q,M_c}(\boldsymbol{c})$$

be the decompressed ciphertext polynomial used by Algorithm 4, where $\boldsymbol{e}_{q,M_c}(\boldsymbol{c}) = \mathrm{Decompress}(\mathrm{Compress}(\boldsymbol{c}, q, M_c), q, M_c) - \boldsymbol{c}$ is the noise caused by the compression. Here, we treat the compression error as an independent polynomial relative to other terms. This is the same type of modeling bridge used in ML-KEM correctness analyses and discussed explicitly in [BKL+25, AAB+24]. This independence is not proven merely by computing the exact marginal, even under the RLWR assumption. During decryption, the algorithm computes

$$\boldsymbol{a} \equiv (x^{n/2} + 1)\boldsymbol{fc}^{\dagger} \pmod{\langle q, x^n + 1\rangle}.$$

Using

$$\boldsymbol{hf} \equiv \boldsymbol{g} \pmod{\langle q, x^n + 1\rangle},$$

and the binary zero-divisor representative of the message lift, the decoding relation uses

$$\boldsymbol{a} \equiv x^{n/2}(x^{n/4} + 1)\boldsymbol{fm} + \boldsymbol{\varepsilon}_{\mathrm{wrap}} \pmod{\langle q, x^n + 1\rangle},$$

where

$$\boldsymbol{\varepsilon}_{\mathrm{wrap}} = (x^{n/2} + 1)\left(\boldsymbol{gs} + \boldsymbol{fe} + \boldsymbol{fe}_{q,M_c}(\boldsymbol{c})\right).$$

Define the integer wrap argument

$$\boldsymbol{z}_{\mathrm{wrap}} = x^{n/2}(x^{n/4} + 1)\boldsymbol{fm} + \boldsymbol{\varepsilon}_{\mathrm{wrap}} \in \mathbb{Z}[x]/(x^n + 1),$$

and its coefficient-wise nearest-q quotient

$$\boldsymbol{e}_{\mathrm{wrap}} = \left\lfloor \frac{1}{q} \cdot \boldsymbol{z}_{\mathrm{wrap}} \right\rceil \in \mathbb{Z}[x]/(x^n + 1).$$

Then, after centered lifting, multiplying by $\boldsymbol{f}_{\mathrm{inv}}$, and reducing modulo $\langle 2, x^{n/2} + 1\rangle$, the intermediate value $\boldsymbol{m}'$ satisfies

$$\boldsymbol{m}' \equiv (x^{n/4} + 1)\boldsymbol{m} + \boldsymbol{e}_{\mathrm{wrap}}\boldsymbol{f}_{\mathrm{inv}} \pmod{\langle 2, x^{n/2} + 1\rangle}.$$

Consequently, if the induced binary wrap error from $\boldsymbol{e}_{\mathrm{wrap}}$ is decoded correctly by the DAWN.SimpleDecoding wrap decoder associated with $x^{n/4} + 1$, the decryption outputs $\boldsymbol{m}$. However, the estimate of the DFR in DAWN assumes independence among the coefficients of $\boldsymbol{z}_{\mathrm{wrap}}$ and ignores the correlation induced by the convolution. We therefore remove this heuristic and modify SimpleDecoding as shown in Algorithm 6.

We use group-level error correction and do not model the coefficients of $\boldsymbol{z}_{\mathrm{wrap}}$ as independent. The group split follows the same coefficient-grouping idea as the multidimensional DFR analysis of Paiva et al. [PSH+25]; Lemma 1 of that work gives the analogous block product decomposition in their compact ML-KEM setting. We found that error correction via zero-divisor encoding could fit such group-level analysis well, and we use this idea with

Algorithm 6 DAWN′.SimpleDecoding($\boldsymbol{a}, \boldsymbol{m}', \boldsymbol{f}_{\text{inv}}$)

Input: Polynomials $\boldsymbol{a}$, $\boldsymbol{m}'$, $\boldsymbol{f}_{\text{inv}}$
Output: Polynomial $\boldsymbol{m}$

$\boldsymbol{\delta} \leftarrow \boldsymbol{a} \pmod{\langle 2, x^{n/4}+1\rangle}$
$\boldsymbol{\eta} \leftarrow 0$
for $i = 0$ to $n/4-1$ **do**
  $v_0 \leftarrow \min\big(\big||\boldsymbol{a}_{[i]}| - \frac{q-1}{2}\big|, \big||\boldsymbol{a}_{[i+n/2]}| - \frac{q-1}{2}\big|\big)$
  $v_1 \leftarrow \min\big(\big||\boldsymbol{a}_{[i+n/4]}| - \frac{q-1}{2}\big|, \big||\boldsymbol{a}_{[i+3n/4]}| - \frac{q-1}{2}\big|\big)$
  if $v_0 < v_1$ **then**
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + x^i \boldsymbol{\delta}_{[i]}$
  else
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + x^{i+n/4} \boldsymbol{\delta}_{[i]}$
  end if
end for
$\boldsymbol{m} \leftarrow \big(\boldsymbol{m}' + \boldsymbol{\eta}\boldsymbol{f}_{\text{inv}} \pmod{\langle 2, x^{n/2}+1\rangle}\big) \pmod{\langle 2, x^{n/4}\rangle}$
return $\boldsymbol{m}$

$\mu = 4$ below. Set $\nu = \frac{n}{4}, Y = x^\nu = x^{n/4}$. Since the ambient ring is $\mathbb{Z}[x]/(x^n+1)$, we have $Y^4 = x^n = -1$. Thus each quarter-cycle group is naturally represented in the small negacyclic ring $\mathbb{Z}[Y]/(Y^4+1)$. For a polynomial $\boldsymbol{a} \in \mathbb{Z}[x]/(x^n+1)$, define its r-th $\mu = 4$ block by

$$\operatorname{block}_r(\boldsymbol{a}) = \boldsymbol{a}_{[r]} + \boldsymbol{a}_{[r+\nu]} Y + \boldsymbol{a}_{[r+2\nu]} Y^2 + \boldsymbol{a}_{[r+3\nu]} Y^3 \in \mathbb{Z}[Y]/(Y^4+1), \qquad 0 \le r < \nu.$$

The fixed polynomials of DAWN′ become particularly simple inside $\mathbb{Z}[Y]/(Y^4+1)$:

$$x^{n/2}+1 = Y^2+1, \qquad x^{n/4}+1 = Y+1.$$

We write $\tau = 1+Y^2, \omega = 1+Y$. Hence $x^{n/2}+1$ and $x^{n/4}+1$ are not treated as generic sparse multipliers; they are part of the four-dimensional algebraic block structure. For two polynomials

$$\boldsymbol{a} = \sum_{u=0}^{\nu-1} x^u \operatorname{block}_u(\boldsymbol{a}), \qquad \boldsymbol{b} = \sum_{u=0}^{\nu-1} x^u \operatorname{block}_u(\boldsymbol{b}),$$

their product block satisfies

$$\operatorname{block}_r(\boldsymbol{a}\boldsymbol{b}) = \sum_{u=0}^{\nu-1} Y^{\epsilon_r(u)} \operatorname{block}_u(\boldsymbol{a}) \operatorname{block}_{(r-u) \bmod \nu}(\boldsymbol{b}),$$

where the carry exponent is

$$\epsilon_r(u) = \begin{cases} 0, & u \le r, \\ 1, & u > r. \end{cases}$$

This identity is the $\mu = 4$ negacyclic group splitting used in the DFR calculation. Let

$$\boldsymbol{U} = \boldsymbol{e} + \boldsymbol{e}_{q,M_c}(\boldsymbol{c}).$$

Then the r-th four-dimensional block of the wrap argument can be written as

$$\operatorname{block}_r(\boldsymbol{z}_{\text{wrap}}) = \sum_{u=0}^{\nu-1} Y^{\epsilon_r(u)} \left[\tau G_u S_v + F_u\big(\tau U_v + Y^2 \omega M_v\big)\right], \qquad v = (r-u) \bmod \nu,$$

where

$$G_u = \operatorname{block}_u(\boldsymbol{g}),\ S_v = \operatorname{block}_v(\boldsymbol{s}),\ F_u = \operatorname{block}_u(\boldsymbol{f}),\ U_v = \operatorname{block}_v(\boldsymbol{U}),\ M_v = \operatorname{block}_v(\boldsymbol{m}).$$

This expression preserves the dependence between the four coefficients in the same quarter-cycle group and also preserves the correlation caused by the shared factor $\boldsymbol{f}$ in $\boldsymbol{fU}$ and $\boldsymbol{fm}$. For each group r, write

$$\boldsymbol{Z}_r = (Z_{r,0}, Z_{r,1}, Z_{r,2}, Z_{r,3}).$$

Define the coefficient-wise wrap quotient

$$K_{r,i} = \left\lfloor \frac{Z_{r,i}}{q} \right\rceil, \qquad i \in \{0,1,2,3\},$$

and the binary wrap bits

$$B_{r,i} \equiv K_{r,i} \pmod 2,$$

modulo $\langle 2, x^{n/4}+1\rangle$, we have $Y = 1$, and therefore the publicly exposed wrap syndrome of group r is

$$W_r = B_{r,0} \oplus B_{r,1} \oplus B_{r,2} \oplus B_{r,3}.$$

However, the message recovery is performed modulo $\langle 2, x^{n/2}+1\rangle$, where $Y^2 = 1$. Thus the two target half-cycle parities are

$$T_{r,0} = B_{r,0} \oplus B_{r,2}, \qquad T_{r,1} = B_{r,1} \oplus B_{r,3}.$$

They satisfy $W_r = T_{r,0} \oplus T_{r,1}$. The DAWN′.SimpleDecoding decoder for group r proceeds as follows. If $W_r = 0$, it outputs $(\widehat{T}_{r,0}, \widehat{T}_{r,1}) = (0,0)$. If $W_r = 1$, it compares the two half-cycles using centered residues. Define the odd-wrap score

$$D_{r,i} = \min_{\ell \in 2\mathbb{Z}+1} |Z_{r,i} - \ell q|, \qquad i \in \{0,1,2,3\}.$$

Then set

$$S_{r,0} = \min(D_{r,0}, D_{r,2}), \qquad S_{r,1} = \min(D_{r,1}, D_{r,3}).$$

If $S_{r,0} < S_{r,1}$, the decoder outputs $(\widehat{T}_{r,0}, \widehat{T}_{r,1}) = (1,0)$; if $S_{r,1} < S_{r,0}$, it outputs $(\widehat{T}_{r,0}, \widehat{T}_{r,1}) = (0,1)$. Ties are counted as failures in the upper-bound estimate. Define the exact bad event for group r by

$$\mathsf{Bad}_r = \{W_r = 0 \ \wedge \ (T_{r,0}, T_{r,1}) \neq (0,0)\} \cup \left\{W_r = 1 \ \wedge \ (\widehat{T}_{r,0}, \widehat{T}_{r,1}) \neq (T_{r,0}, T_{r,1})\right\}.$$

Equivalently, Bad_r is the event that the exposed $\langle 2, x^{n/4}+1\rangle$-syndrome and the $\mathbb{Z}_q$-centered residue test do not recover the correct $\langle 2, x^{n/2}+1\rangle$-wrap representative for group r. Let

$$P_r^{(4)} = \mathsf{Law}(\boldsymbol{Z}_r)$$

be the exact four-dimensional joint distribution of the group. The group failure probability is

$$p_r^{(4)} = \sum_{\boldsymbol{z} \in \mathbb{Z}^4} P_r^{(4)}(\boldsymbol{z}) \cdot \mathbf{1}_{\mathsf{Bad}_r}(\boldsymbol{z}).$$

In this definition, no independence is used between the four coordinates of $\boldsymbol{Z}_r$. Since the distribution is stationary over r, we obtain the following inequality by a union bound over the $\nu = n/4$ groups,

$$\mathsf{DFR}_{\mathrm{DAWN}'} \leq \sum_{r=0}^{\nu-1} p_r^{(4)} = \nu p_0^{(4)}.$$

In practice, we compute the DFR using the projection method, following [PSH+25], because directly evaluating the four-dimensional joint distribution is computationally expensive.

Let $\mathcal{P}_{\text{cross}} = \{(0,1),(0,3),(2,1),(2,3)\}$; these are precisely the pairs crossing the two half-cycles $\{0,2\}$ and $\{1,3\}$. The event Bad_r is conservatively bounded by cross-half wrap collisions and cross-half centered residue confusions. For $(i,j) \in \mathcal{P}_{\text{cross}}$, we use

$$\Pr[|Z_{r,i}| + |Z_{r,j}| \geq q] \leq \sum_{\sigma,\tau\in\{\pm 1\}} \Pr[\langle \sigma \boldsymbol{e}_i + \tau \boldsymbol{e}_j, \boldsymbol{Z}_r \rangle \geq q] .$$

Then, we obtain an upper bound $\widehat{\mathsf{DFR}}_{\text{DAWN}'}$ of DFR as

$$\widehat{\mathsf{DFR}}_{\text{DAWN}'} = \sum_{r=0}^{\nu-1} \sum_{(i,j)\in\mathcal{P}_{\text{cross}}} \sum_{\sigma,\tau\in\{\pm 1\}} \Pr[\langle \sigma \boldsymbol{e}_i + \tau \boldsymbol{e}_j, \boldsymbol{Z}_r \rangle \geq q] ,$$

where $\boldsymbol{e}_i$ is the i-th standard basis of $\mathbb{Z}^4$.

3.1.2 Flexible Distributions from Bit Operations

To obtain a better trade-off between security and compactness, the original DAWN proposal [LZL+25] uses sparse ternary distributions. However, straightforward sampling from such distributions may require conditional checks or table-based logic, which is not constant-time and may be vulnerable to side-channel attacks. In contrast, ML-KEM [Nat24a] uses centered binomial distributions, which can be sampled by additions and subtractions of random bits. For compact DAWN-type encryption with a small modulus $q = 769$, however, centered binomial distributions may have variance that is too large. We therefore also allow simple products of random bits, which provide finer control over the variance while preserving a simple and branchless sampling structure.

Consider sparse ternary distributions of the form

$$\mathcal{S}_r = \{-1 : r, \quad 1 : r, \quad 0 : 1 - 2r\}.$$

A useful family of such distributions can be sampled using only subtractions and multiplications of independent random bits. Let

$$r = \frac{2^{l_0} - 1}{2^{2l_0+l_1}}, \qquad l_0 \in \mathbb{Z}_{\geq 1}, \quad l_1 \in \mathbb{Z}_{\geq 0}.$$

Let U_i be independent Bernoulli random variables with parameter $1/2$, and define

$$A = \prod_{i=0}^{l_0-1} U_i, \qquad B = \prod_{i=l_0}^{2l_0-1} U_i, \qquad C = \prod_{i=2l_0}^{2l_0+l_1-1} U_i,$$

where the empty product is interpreted as 1. Then

$$X = (A - B)C$$

satisfies

$$\Pr[X = 1] = \Pr[X = -1] = \frac{2^{l_0} - 1}{2^{2l_0+l_1}},$$

and hence $X \sim \mathcal{S}_r$. Such distributions require $2l_0 + l_1$ bits to sample one element. The centered binomial distribution with parameter 1 is identified as the special case $l_0 = 1, l_1 = 0$. For DAWN′, we use the following sparse ternary distributions:

$$\mathcal{S}_{1/4} = U_0 - U_1, \qquad \mathcal{S}_{1/8} = (U_0 - U_1)U_2, \qquad \mathcal{S}_{3/16} = U_0U_1 - U_2U_3.$$

These distributions are therefore sampled using only simple bit operations. In our practical parameter selections, we use sparse ternary distributions, centered binomial distributions, and their combinations to achieve fine-grained control over the noise scale.

3.2 Constant-time Compression

In DAWN, the error-correction mechanism enables aggressive compression, a key contributor to the ciphertext's compactness. In the original DAWN construction [LZL+25], compression is described as division with rounding, namely the map $a \mapsto \lfloor a/d_c \rceil$, with the corresponding decompression given by $b \mapsto d_c b$, where d_c is the scale of ciphertext compression. However, division on the Cortex-M4 platform is not constant-time. To avoid this issue, we replace division by fixed-point multiplication and shifts, while preserving a similar approximation. Moreover, on Cortex-M4, 32-bit multiplication incurs a cost comparable to addition, allowing us to tune the compression procedure without sacrificing efficiency.

Formally, to compress an element from $\mathbb{Z}_q$ to $\mathbb{Z}_{M_c}$, we fix the following scaling factors with ℓ bits of precision:

$$\delta_e = \left\lfloor \frac{M_c}{q} \cdot 2^{\ell} \right\rfloor, \qquad \delta_d = \left\lfloor \frac{q}{M_c} \cdot 2^{\ell} \right\rfloor.$$

The compression map is then defined as

$$\mathrm{Compress}(\boldsymbol{a}, q, M_c) := \sum_{i=0}^{n-1} \left(\left(\boldsymbol{a}_{[i]} \cdot \delta_e \right) \gg \ell \right) x^i,$$

and the corresponding decompression map is

$$\mathrm{Decompress}(\boldsymbol{a}, q, M_c) := \sum_{i=0}^{n-1} \left(\left((\boldsymbol{a}_{[i]} \cdot \delta_d) \gg \ell \right) + \lceil \frac{q}{2M_c} \rceil \right) x^i.$$

We add the constant $\lceil q/(2M_c) \rceil$ to make the compression noise symmetric around zero, thus lowering the DFR. For the recommended DAWN′ parameter sets, we choose $\ell = 16$. By setting $\ell = 16$, we can utilize the 16-bit multiplication like `smul{b,t}{b,t}`, `smulw{b,t}` and packing instructions `pkhbt`, `pkhtb` in Cortex-M4 for efficient compress and decompress implementations. The resulting values of δ_e and δ_d are listed in Table 2.

Table 1: Comparison of DFRs in DAWN and DAWN′

scheme	DAWN-α-512	DAWN-β-512	DAWN′-128	DAWN-α-1024	DAWN-β-1024	DAWN′-256
$\log_2(\mathrm{DFR}_{\mathrm{heuristic}})$	-133	-130	-142	-163	-138	-189
$\log_2(\widehat{\mathrm{DFR}}_{\mathrm{DAWN'}})$	-120	-106	-131	-150	-115	-175

Table 2: DAWN′ parameters

scheme	n	q	(D_g, D_f, D_s, D_e)	M_c	δ_e	δ_d
DAWN′-128	512	769	$(\mathcal{S}_{1/4}, \mathcal{S}_{3/16}, \mathcal{S}_{1/4}, \mathcal{S}_{1/4})$	147	12528	342838
DAWN′-256	1024	769	$(\mathcal{S}_{3/16}, \mathcal{S}_{1/8}, \mathcal{S}_{1/4}, \mathcal{S}_{1/4})$	256	21817	196864

3.3 Parameter Recommendation of DAWN′

We present the recommended parameter sets for DAWN′ using the distributions introduced above. Our new analysis shows that the DFR estimates of the original DAWN increase by several orders of magnitude, reaching the security boundary and necessitating a reselection of parameters. To quantify the impact of this analysis without assuming independence in polynomial convolution, we compare the results in Table 1. Specifically, $\mathrm{DFR}_{\mathrm{heuristic}}$ denotes the DFR estimated under the independence assumption, whereas $\widehat{\mathrm{DFR}}_{\mathrm{DAWN'}}$ denotes the DFR obtained with our proposed approach.

Consistent with the original DAWN design, we provide two parameter sets targeting 128-bit and 256-bit classical security, corresponding to the NIST-I and NIST-V security levels. Secret-key recovery relies on the hardness of the NTRU problem, whereas plaintext recovery relies on the hardness of the RLWE problem. The resulting parameters are listed in Table 2. This parameter reselection applies milder ciphertext compression, yielding reliable lower DFR estimates at the cost of only an approximately 5% increase in ciphertext length, as shown in Table 4.

We use the lattice estimator[1] of [APS15] to derive concrete security estimates for DAWN′. We run the estimator under the default BKZ model `MATZOV`, and the resulting concrete security estimates are reported in Table 3.

Table 3: Concrete Security of DAWN′ Against Different Attacks

scheme	NTRU			RLWE		
	attack	classical	quantum	attack	classical	quantum
DAWN′-128	`primal usvp`	146.0	140.0	`primal usvp`	149.3	142.9
	`primal bdd`	142.0	135.9	`primal bdd`	145.3	138.9
	`primal hybrid`	142.0	135.9	`dual hybrid`	141.7	136.1
DAWN′-256	`primal usvp`	277.0	257.9	`primal usvp`	297.2	276.0
	`primal bdd`	272.0	252.9	`primal bdd`	292.1	271.0
	`primal hybrid`	272.0	252.9	`dual hybrid`	280.9	263.8

4 Optimized DAWN′ Implementation on ARM Cortex-M4

This section describes the proposed optimizations for two core arithmetic components of DAWN′: arithmetic over R_q and R_2.

4.1 DAWN′ Optimizations for R_q

The small modulus of DAWN′, $q = 769$, makes it well suited for efficient implementation on the ARM Cortex-M4 using improved Plantard arithmetic [HZZ+22, HAZ+24]. Huang et al. [HAZ+24] have previously studied highly optimized small NTT implementations modulo 769 in the context of Dilithium. Although their implementation is already highly optimized, DAWN′ uses higher-degree polynomials, with $n \in \{512, 1024\}$. This makes base multiplication and inversion more challenging than in schemes such as Kyber and Dilithium, which operate on length-256 polynomials.

In this section, we extend the implementation technique of [HAZ+24] to DAWN′. We show that improved Plantard arithmetic is also well suited to incomplete, high-degree NTTs, as well as to the associated base multiplication and inversion operations.

4.1.1 NTT and INTT Implementations

Following its RLWE-based design, the parameter sets of DAWN′ share the same modulus $q = 769$ and differ only in the polynomial degree, namely $n \in \{512, 1024\}$. For NTT and INTT over $\mathbb{Z}_{769}$, we build on the optimized implementation of [HAZ+24], which employs the improved Plantard arithmetic of [HZZ+22] to accelerate modular multiplications by twiddle factors in NTT and INTT. The use of improved Plantard arithmetic also helps eliminate redundant modular reductions in the NTT/INTT computation.

Moreover, the NTT and INTT implementations in [HAZ+24] employ a $4 + 3$ layer-merging strategy based on the CT butterfly unit, which reduces memory traffic by keeping intermediate values in registers across multiple butterfly layers. Since all DAWN′ parameter

[1] https://github.com/malb/lattice-estimator/, with the latest commit `3e48ef4`

sets use the same modulus, the same arithmetic kernels can be reused across different transform sizes with only minor adaptations. In particular, supporting different polynomial degrees only requires adjusting the butterfly strides and loop bounds, which reduces the implementation effort across security levels.

4.1.2 Optimizing the Degree-k Base Multiplication

To improve the performance of base multiplication, we utilize the lazy reduction technique first introduced in [ABCG20] to substantially reduce the number of modular reductions required for each coefficient. Note that the optimized Kyber base multiplication in [ABCG20] corresponds to the special case $k = 2$. We generalize this technique to larger k and conduct range analysis for high-degree base multiplication. We also present a register-reshuffling strategy to further reduce the number of multiplications in high-degree base multiplication.

Lazy reduction. Recall from Equation 1 that the computation of each coefficient consists of two parts: the first part is not multiplied by the constant r, while the second part requires an additional multiplication by r. For the first part, instead of computing each modular product $(\hat{a}_{[j]}\hat{b}_{[i-j]} \bmod q)$ separately and then summing the reduced values, lazy reduction first accumulates the intermediate products with SIMD instructions like smla{b,t}{b,t}, smuadx, smladx, and performs modular reduction on the accumulated results if they do not overflow or exceed the input range of the modular reduction.

For the second part that contains only one product of $\hat{a}_{[j]}$ and $\hat{b}_{[i+k-j]}$, the lazy reduction is similar to the case in Kyber [ABCG20]. The total number of modular reductions for this coefficient can be reduced from $k+1$ to 2. One of them can be performed with the efficient 2-cycle Plantard multiplication by the twiddle factor. On the other hand, for coefficients whose second part contains multiple products $\hat{a}_{[j]}\hat{b}_{[i+k-j]}$, we need to accumulate these products and compute modular reduction first. Then, the product of the accumulated results and the twiddle factor r is computed with the 2-cycle Plantard multiplication by the twiddle factor. Together with the reduction from the first part, this brings the total to 3, rather than the $k+1$ modular multiplications a naive schedule would require.

Range analysis of base multiplication. Due to the small 10-bit modulus used in DAWN′ and the use of Plantard arithmetic, all intermediate values arising in base multiplication remain within the valid input range of Plantard arithmetic, even for the largest base multiplication used in DAWN′, where $k = 8$.

The worst-case accumulation occurs when one coefficient accumulates k products of the form $\hat{a}_{[j]}\hat{b}_{[i-j]}$. Since the output of the 7-layer NTT with Plantard arithmetic is bounded by $q + 3.5q = 4.5q$, the accumulated value is bounded by $k \times (4.5q)^2$. For the largest case $k = 8$, this gives $8 \times (4.5q)^2 = 162q^2 < 2^{28}$. This bound is well within the range of a 32-bit signed integer and also lies within the input range of Plantard arithmetic [HZZ+24]. Therefore, base multiplication in DAWN′ can safely employ lazy reduction extensively, thereby substantially reducing the number of modular reductions.

Reducing the number of multiplications with smuad{x}, smlad{x}. In addition to applying lazy reduction to reduce the number of modular reductions, we further reduce the number of multiplication instructions by fully exploiting the dual 16-bit multiply and multiply-accumulate instructions smuad, smuadx, smlad, and smladx. The key idea is to shuffle the register layout so that two coefficient products contributing to the same output coefficient can be computed by one dual multiplication instruction. We illustrate the proposed method for $k = 8$. The original mapping between registers and coefficients is shown in Figure 1. Each register stores two adjacent 16-bit coefficients; that is, register r_i stores $\hat{a}_{[2i]}$ and $\hat{a}_{[2i+1]}$ and register r_{4+i} stores $\hat{b}_{[2i]}$ and $\hat{b}_{[2i+1]}$, for $i \in \{0, 1, 2, 3\}$.

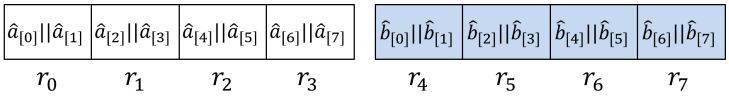


Figure 1: The original mapping of the registers and coefficients with $k = 8$.

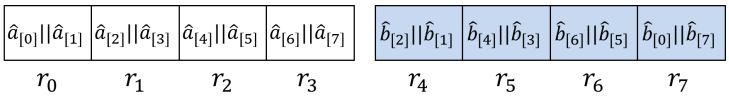


Figure 2: Register reshuffling for computing even-indexed coefficients with $k = 8$.

Odd-indexed coefficients. We separate the computation according to the parity of i. We find that the odd-indexed coefficients can be computed efficiently using smuadx and smladx instructions with the original register layout. Since these dual 16-bit instructions multiply cross halfwords, this saves one multiplication instruction per pair. For example, instruction smuadx t, r_0, r_4 computes two multiplications $t = \hat{a}_{[0]}\hat{b}_{[1]} + \hat{a}_{[1]}\hat{b}_{[0]}$, which contributes to the odd-indexed coefficient $\hat{c}_{[1]}$. More specifically, when $k = 8$, the coefficient $\hat{c}_{[1]}$ is given by:

$$\hat{c}_{[1]} = \left(\hat{a}_{[0]}\hat{b}_{[1]} + \hat{a}_{[1]}\hat{b}_{[0]} + r\left(\hat{a}_{[2]}\hat{b}_{[7]} + \hat{a}_{[3]}\hat{b}_{[6]} + \hat{a}_{[4]}\hat{b}_{[5]} + \hat{a}_{[5]}\hat{b}_{[4]} + \hat{a}_{[6]}\hat{b}_{[3]} + \hat{a}_{[7]}\hat{b}_{[2]}\right)\right) \bmod q.$$

The eight products in this coefficient can be grouped into four cross-halfword pairs. Using the original register layout, each pair can be computed or accumulated using one smuadx or smladx instruction. Compared with scalar halfword multiplication, this reduces the number of multiplication instructions from k to $k/2$ for each odd-indexed coefficient. Since there are $k/2$ odd-indexed coefficients, the original packing saves a total of $\frac{k}{2} \cdot \frac{k}{2} = \frac{k^2}{4}$ multiplication instructions for the odd-indexed coefficients in one length-k base multiplication.

Even-indexed coefficients. For the even-indexed coefficients, however, the required source coefficients are misaligned under the original packing. A straightforward implementation would have to use scalar halfword multiplication instructions, such as smulbb, smulbt, smultb, and smultt, each of which computes only one product. For larger-degree base multiplications, this substantially increases the number of multiplication instructions. Specifically, for $k/2$ even-indexed coefficients, using the original register layout requires $\frac{k^2}{2}$ multiplications for multiplying $\hat{a}_{[j]}$ and $\hat{b}_{[i+k-j]}$.

Algorithm 7 Register reshuffling for computing even-index product with $k = 8$

Input: Registers: $r_4 = \hat{b}_{[0]}||\hat{b}_{[1]}, r_5 = \hat{b}_{[2]}||\hat{b}_{[3]}, r_6 = \hat{b}_{[4]}||\hat{b}_{[5]}, r_7 = \hat{b}_{[6]}||\hat{b}_{[7]}$.
Output: Registers: $t = \hat{b}_{[2]}||\hat{b}_{[1]}, r_5 = \hat{b}_{[4]}||\hat{b}_{[3]}, r_6 = \hat{b}_{[6]}||\hat{b}_{[5]}, r_7 = \hat{b}_{[0]}||\hat{b}_{[7]}$.
1: pkhbt t, r_5, r_4;
2: pkhbt r_5, r_6, r_5;
3: pkhbt r_6, r_7, r_6;
4: pkhbt r_7, r_4, r_7;
5: **return** t, r_5, r_6, r_7.

To minimize this overhead, we follow a similar approach in [KMRV18, KRS19] to reshuffle the register layout of one input polynomial using $k/2$ pkhbt packing instructions, as shown in Algorithm 7. Unlike their schoolbook multiplication, we specialize the register layout to the twisted structure $\mathbb{Z}_q[x]/(x^k - r)$ and integrate it with a Plantard-based lazy-reduction schedule. We further derive exact instruction counts that quantify the resulting savings as a function of k as follows. The resulting layout is shown in Figure 2. After this

reshuffling, the operands required for the even-indexed coefficients have been aligned so that smuad and smlad can be used to compute or accumulate two products contributing to the same even-indexed coefficient. This reduces the number of multiplication instructions at the cost of only a small number of register-packing instructions.

Specifically, there are k products of $\hat{a}$ and $\hat{b}$ during the computations of each coefficient. For the even-indexed $\hat{c}_{[0]}$ for $k = 8$, there are only $\frac{k-2}{2}$ pairs of products that can be merged using smuad or smlad due to the twisted structure:

$$\hat{c}_{[0]} = \left(\hat{a}_{[0]}\hat{b}_{[0]} + r\left(\hat{a}_{[1]}\hat{b}_{[7]} + \hat{a}_{[2]}\hat{b}_{[6]} + \hat{a}_{[3]}\hat{b}_{[5]} + \hat{a}_{[4]}\hat{b}_{[4]} + \hat{a}_{[5]}\hat{b}_{[3]} + \hat{a}_{[6]}\hat{b}_{[2]} + \hat{a}_{[7]}\hat{b}_{[1]}\right)\right) \bmod q.$$

Therefore, $\frac{k-2}{2}$ multiplication instructions are saved per even-indexed coefficient, and $\frac{k-2}{2} \times \frac{k}{2}$ multiplication instructions are saved across all even-indexed coefficients in length-k base multiplication. Since the reshuffling requires only $\frac{k}{2}$ packing instructions, the saving is $\frac{k-2}{2} \times \frac{k}{2} - \frac{k}{2}$ instructions per length-k base multiplication.

Together with the odd-indexed saving, for a complete polynomial multiplication with 128 such base multiplications, using the dual multiplication instructions on Cortex-M4 helps to reduce multiplications by

$$128 \times \left(\frac{k^2}{4} + \frac{k-2}{2} \times \frac{k}{2} - \frac{k}{2}\right) = 128 \times \left(\frac{k^2}{4} + \frac{k}{2} \times \frac{k-4}{2}\right), k \geq 4.$$

For $k = 4$, the saving from the even-indexed coefficients is fully offset by the reshuffling cost, and hence the net gain for the even-indexed part is zero. For $k > 4$, the saving from the even-indexed coefficients becomes positive and increases with k, leading to larger performance improvements for higher-degree base multiplications.

4.2 Base Inversion

For DAWN′-128 and DAWN′-256, where $k = 4$ and $k = 8$, respectively, the degree of each base polynomial is sufficiently small that all input coefficients and intermediate values can be retained in registers. We therefore implement base inversion using a direct matrix-based schoolbook method. More precisely, let

$$a(x) = \sum_{i=0}^{k-1} a_{[i]}x^i \in \mathbb{Z}_q[x]/(x^k - r).$$

Multiplication by $a(x)$ defines a linear transformation over $\mathbb{Z}_q^k$. For example, when $k = 4$, this transformation is represented with respect to the basis $\{1, x, x^2, x^3\}$ by

$$M_a = \begin{pmatrix} a_{[0]} & ra_{[3]} & ra_{[2]} & ra_{[1]} \\ a_{[1]} & a_{[0]} & ra_{[3]} & ra_{[2]} \\ a_{[2]} & a_{[1]} & a_{[0]} & ra_{[3]} \\ a_{[3]} & a_{[2]} & a_{[1]} & a_{[0]} \end{pmatrix},$$

where the wrapped terms follow from the relation $x^4 = r$. Hence, $a(x)$ is invertible if and only if $\det(M_a) \neq 0 \pmod q$. Writing

$$a(x)^{-1} = \sum_{i=0}^{k-1} b_{[i]}x^i,$$

the coefficient vector of the inverse is obtained as

$$\begin{pmatrix} b_{[0]} \\ b_{[1]} \\ \vdots \\ b_{[k-1]} \end{pmatrix} = M_a^{-1} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \det(M_a)^{-1} \operatorname{adj}(M_a) \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

In the implementation, the required cofactors and the determinant are symbolically expanded in advance and evaluated using fixed sequences of schoolbook multiplications and additions. Consequently, only one inversion in $\mathbb{Z}_q$ is required. Thereafter, the resulting cofactors are multiplied by $\det(M_a)^{-1}$. The same construction is applied to the $k = 8$ case using the corresponding 8×8 multiplication matrix. Compared with generic Gaussian elimination or polynomial extended Euclidean algorithms, this fully unrolled formulation avoids data-dependent control flow and indexing overhead and is particularly efficient for these small fixed dimensions on small IoT devices.

As in base multiplication, we employ lazy reduction to accumulate intermediate products before modular reduction whenever the derived range bounds permit. Moreover, multiplications by fixed constants and powers of the associated twiddle factor are implemented using efficient two-cycle Plantard multiplication by a constant, further reducing the cost of base inversion.

4.3 DAWN′ Optimizations for R_2

Due to the introduction of the zero-divisor encoding technique, DAWN′ also requires binary polynomial multiplication modulo $\langle 2, x^m + 1\rangle$ for various polynomial degrees $m = 2, 4, \ldots, n/2$. These binary polynomial multiplications are widely used in FastInversion (Algorithm 1), Step 5 of Algorithm 5, and simple decoding (Algorithm 6) of DAWN′.

Previous optimizations for polynomial multiplication over R_2 mainly target code-based schemes, such as Classic McEliece [CC21], BIKE [CCK21, CGKT22], and HQC [LJKH25]. However, polynomial multiplications over R_2 in these code-based schemes are typically performed on very high-degree polynomials. For example, the smallest HQC parameter set HQC-1 uses $n = 17669$, and the smallest BIKE polynomial-multiplication size is $r = 12323$. As a result, existing optimizations based on Karatsuba, Toom–Cook, or additive FFT are designed for medium- or large-degree polynomials and cannot be directly transferred to the small-degree binary polynomial multiplications required in DAWN′. Hence, we tailor specific optimizations for DAWN′'s small-degree polynomials on ARM Cortex-M4.

4.3.1 Radix-16 Representation for Polynomial Multiplication in R_2

Let $R_2 = \mathbb{F}_2[x]/(x^m + 1)$, and let a polynomial $\mathbf{a} \in R_2$ be written as $\mathbf{a} = a_{[0]} + a_{[1]}x + \cdots + a_{[m-1]}x^{m-1}$, with $a_{[i]} \in \mathbb{F}_2$. Since the coefficient field has characteristic two, polynomial addition and subtraction are both implemented by coefficient-wise `eor` operations. The most straightforward way to implement polynomial multiplication in R_2 is the schoolbook method using `and` for coefficient multiplications and `eor` for coefficient additions. A direct cyclic schoolbook multiplication requires m^2 `and` operations and $m(m-1)$ `eor` operations. As m in DAWN′-256 can be as large as 512, this quadratic complexity makes binary polynomial multiplication a significant performance bottleneck.

To reduce the cost of polynomial multiplication over R_2, we revisit existing optimizations for binary polynomial arithmetic in code-based cryptography. In particular, we find that the radix-16 representation from [CC21] is also well suited to the small-degree polynomial multiplication over R_2 on ARM Cortex-M4. The main idea is to store each binary coefficient in one 4-bit lane. Thus, eight coefficients can be packed into a single 32-bit word.

Smaller-degree polynomials in R_2. For a length-8 polynomial $a(x) = a_{[0]} + a_{[1]}x + \cdots + a_{[7]}x^7$, its radix-16 representation is $\widetilde{a} = a_{[0]} + a_{[1]}2^4 + \cdots + a_{[7]}2^{28} = \sum_{i=0}^{7} a_{[i]}2^{4i}$. Similarly, $\widetilde{b} = \sum_{i=0}^{7} b_{[i]}2^{4i}$. Multiplying the two packed words gives

$$\widetilde{p} = \widetilde{a}\widetilde{b} = \sum_{\ell=0}^{14} \left(\sum_{i+j=\ell} a_{[i]}b_{[j]} \right) 2^{4\ell}.$$

Even in the worst case where all coefficients of a and b are one, each coefficient of the integer convolution is at most 8, namely $\sum_{i+j=\ell} a_{[i]}b_{[j]} \leq 8 < 2^4$, so no carry propagates between adjacent 4-bit lanes. Consequently, the least significant bit of each 4-bit lane gives the corresponding coefficient modulo 2.

The reduction modulo x^8+1 is also simple, since over $\mathbb{F}_2$ we have $x^8 \equiv 1$. Let $\widetilde{p}_{\text{lo}}$ and $\widetilde{p}_{\text{hi}}$ denote the lower and upper 32-bit halves of the 64-bit product, and let $M = \texttt{0x11111111}$ be the mask selecting the least significant bit of each 4-bit lane. Then the product in $\mathbb{F}_2[x]/(x^8+1)$ is represented by $\widetilde{c} = (\widetilde{p}_{\text{lo}} \,\&\, M) \oplus (\widetilde{p}_{\text{hi}} \,\&\, M) = (\widetilde{p}_{\text{lo}} \oplus \widetilde{p}_{\text{hi}}) \,\&\, M$. On the ARM Cortex-M4, this length-8 multiplication can therefore be implemented using one umull, one and, and one eor instruction. Compared with the direct schoolbook implementation, which requires 64 and operations and 56 eor operations for one length-8 cyclic multiplication, the radix-16 representation substantially reduces the instruction count, providing significant speedups for polynomial multiplication over R_2. For the smaller degrees $m = 2$ and $m = 4$, the packed radix-16 representation occupies only the lower 8 and 16 bits of a 32-bit register, respectively. Therefore, the integer product can be computed using smulbb, which multiplies the lower 16-bit halfwords of two registers and returns a 32-bit result.

Higher-degree polynomials in R_2 As for higher-degrees, we represent a polynomial $\mathbf{a}$ in R_2 with radix-16 as follows: $\mathbf{A} = (A_0, \ldots, A_{d/8-1})$ representing $A(x) \in \mathbb{F}_2[x]/(x^d+1)$ for a specific degree d, where each $A_i = \sum_{j=0}^{7} a_{[8i+j]}2^{4j}$ for $i \in [0, d/8-1], j \in [0, 7]$. The number of words required to store the polynomial a is $d/8$. Depending on the size of $d/8$, we utilize different implementation strategies for polynomial multiplication modulo $\langle 2, x^d+1\rangle$.

For $d \leq 64$, we adopt product-scanning strategy since we can utilize the umlal instruction to accumulate the products with the same index. Note that we need to mask and get the least significant bit before using umlal to avoid overflow. For $d > 64$, we utilize Karatsuba's divide-and-conquer method to recursively split the high-degree polynomials into small 4-word product-scanning multiplications. Steps 5 and 6 of Algorithm 5 contains asymmetric polynomial multiplications modulo $x^{n/2}+1$, with operand lengths $n/4$ and $n/2$. We therefore design a dedicated multiplication routine for this operand shape, eliminating the redundant operations that would arise from zero-padding the shorter operand and invoking a generic and slower length-$n/2$ multiplier.

4.3.2 Fast Inversion with Radix-16 Representation

The fast inversion procedure, shown in Algorithm 1, is a core operation in the PKE key generation of DAWN′. It utilizes the mathematical property, i.e., $(x+1)^{n/4} = x^{n/4}+1$ over $\mathbb{F}_2$, to efficiently compute inversion over $\langle 2, x^{n/4}+1\rangle$. This procedure involves not only polynomial multiplications modulo $\langle 2, x^m+1\rangle$ for various degrees $m = 2^i, i \in [0, \log_2(n/4)]$, but also routines like exact polynomial divisions by x^m+1. Therefore, to fully exploit the radix-16 representation for polynomial multiplication over R_2, we need to re-design efficient routines for fast inversion under radix-16 representation.

Minimizing radix conversion. Previous implementations for code-based schemes usually store binary polynomials as packed bit strings. Using a radix-16 representation in such settings would introduce non-negligible conversions between the bit-string and radix-16 formats [SKLE25]. Moreover, since radix-16 stores each binary coefficient in a 4-bit lane, it increases the memory footprint by a factor of four compared with a packed bit-string representation.

In DAWN′, however, this overhead is largely avoided. The polynomial f in Algorithm 3 is involved in arithmetic over both R_q and R_2, and therefore it must already be stored as an array of 16-bit integers for the R_q-arithmetic. In addition, as the polynomial inversion

is conducted over $\langle 2, x^{n/4}+1\rangle$, the conversion to the radix-16 representation can be merged with the reduction of f modulo $x^{n/4}+1$ immediately after sampling. In this fused step, the inline barrel shifter is used to place the corresponding bits into their radix-16 lanes while folding the coefficients modulo $x^{n/4}+1$, which would not introduce extra cost. We also merge this step with the invertibility check (Step 1 in Algorithm 1) over R_2, further reducing the memory access cost of these steps. In general, only two radix-conversion steps are required: one before Step 9 of Algorithm 3, and one at the end of Algorithm 3 to convert f_{inv} into a bit string.

Polynomial division by $x^m + 1$ with radix-16. For the division $k = \frac{f+1}{x^m+1}$, with $g = f+1, m = 2^i, i \in [0, \log_2(d)]$, this division by x^m+1 can be implemented efficiently using a recurrence that involves only eor operations, where the quotient coefficients are computed as

$$k_t = \begin{cases} g_t, & 0 \le t < m, \\ g_t \oplus k_{t-m}, & m \le t < d. \end{cases}$$

In the radix-16 representation, this operation can be achieved with prefix eor over 4-bit lanes with stride m. As shown in Algorithm 8, for small values of m, it can be implemented using eor instructions combined with an inline barrel shifter. When m is a multiple of 8, the operation is word-aligned and can instead use the faster word-level eor instruction. Because each eor processes eight coefficients packed into a 32-bit word, this implementation requires fewer instructions than the original representation, in which each binary coefficient is stored in a 16-bit word. It therefore achieves substantial performance gains through greater parallelism. Similarly, other fast inversion routines can be implemented in a similar manner. In summary, utilizing the radix-16 representation can bring performance improvement with minimal overhead.

Algorithm 8 Polynomial division by $x^m + 1$ with radix-16

Input: Radix-16 packed polynomial $\mathbf{A} = (A_0, \ldots, A_{d/8-1})$ representing $A(x) \in \mathbb{F}_2[x]/(x^d+1)$, degree d, and step size m
Output: $\mathbf{A}$ is overwritten by $Q(x) = A(x)/(x^m+1)$

if $m \bmod 8 = 0$ **then**
  $w \leftarrow m/8$
  for $i \leftarrow w$ **to** $d/8-1$ **do**
    $A_i \leftarrow A_i \oplus A_{i-w}$
  end for
else
  $P \leftarrow 0$ ▷ Previously computed packed quotient word
  $\delta \leftarrow 4m$ ▷ Distance between dependent bit lanes
  for $i \leftarrow 0$ **to** $d/8-1$ **do**
    $W \leftarrow A_i \oplus \big(P \gg (32-\delta)\big)$
    $\delta' \leftarrow \delta$
    while $\delta' < 32$ **do**
      $W \leftarrow W \oplus (W \ll \delta')$
      $\delta' \leftarrow 2\delta'$
    end while
    $A_i \leftarrow W$
    $P \leftarrow W$
  end for
end if
return A

Table 4: Key size and security comparison of DAWN′ and DAWN.

Schemes	NIST Sec.	Ciphertext (B)	Public key (B)	Sec. level ($\log_2$)
DAWN′-128	NIST-I	461	615	141.7
DAWN-α-512 [LZL+25]	NIST-I	436	615	139.4
DAWN′-256	NIST-V	1024	1229	272.0
DAWN-α-1024 [LZL+25]	NIST-V	973	1229	269.7

4.3.3 Constant-time Implementation

We additionally identify several sources of non-constant-time behavior in the original DAWN implementation [LZL+25], which may be vulnerable to side-channel attacks. As discussed in Subsection 3.2, its compression routine relies on the costly integer division, whose execution time may depend on the operands. Moreover, the original polynomial multiplication over R_2 processes only nonzero coefficients, introducing secret-dependent branches and memory-access patterns. We thoroughly revise these routines to use masking and regular memory accesses, obtaining a constant-time implementation.

5 Experimental Results

This section includes the experimental results of DAWN′ on ARM Cortex-M4. We utilize the state-of-the-art Keccak implementation from [HAZ+24] for the polynomial sampling in the schemes to ensure a fair comparison. The benchmarks for DAWN′ and DAWN were conducted on the NUCLEO-L4R5ZI board, which features 2 MB of Flash and 640 KB of RAM. The experimental results were obtained by setting the clock frequency to 20 MHz. All experiments were compiled with `arm-none-eabi-gcc` version 13.2.1 at the `-O3` optimization level without `-flto`.

5.1 Key Size and Security Comparisons with DAWN

Table 4 compares the key size and concrete security of DAWN and DAWN′ using the state-of-the-art lattice estimator[2] introduced in [APS15]. The results show that DAWN′ preserves the public-key size of DAWN while increasing the ciphertext size by only approximately 5% and providing slightly higher estimated security. Moreover, unlike DAWN, DAWN′ does not rely on the independence assumption of polynomial convolution. Overall, these properties make DAWN′ a more robust refinement of DAWN.

5.2 Performance of Core Operations of DAWN′

Polynomial sampling and arithmetic over R_q. Table 5 reports the performance of polynomial sampling and arithmetic over R_q in DAWN′. Owing to the bit-operation-friendly distributions described in Subsubsection 3.1.2, sampling the polynomials s and e is around 5× faster in DAWN′ than in DAWN. Our Cortex-M4-optimized arithmetic over R_q substantially outperforms both the reference implementation of DAWN′ and the original DAWN implementation [LZL+25]. For DAWN′-128, due to the usage of improved Plantard arithmetic and extensive lazy reduction strategy, the optimized NTT, INTT, base multiplication, and base inversion achieve speedups of 5.72×, 5.86×, 3.90×, and 5.69×, respectively, over their corresponding reference implementations. For DAWN′-256, the NTT, INTT, and base inversion achieve comparable speedups. Its base multiplication attains an even larger speedup of 8.14×, primarily because we further reduce the number of multiplications through extensive lazy reduction and register reshuffling as shown in Subsubsection 4.1.2.

[2] https://github.com/malb/lattice-estimator/, we use the latest commit `3e48ef4`.

Table 5: Performance of core operations over R_q on ARM Cortex-M4 (cycles).

Schemes	NTT	INTT	Basemul	Baseinv	Sample_f	$\text{Sample}_{s,e}$
DAWN′-128 (M4)	9033	9262	6865	25236	37724	14583
DAWN′-128 (ref)	51636	54257	26774	143497	37722	14581
DAWN-α-512 [LZL+25]	51637	53260	28692	140939	39588	70212
DAWN′-256 (M4)	17737	18051	18131	38295	58734	27855
DAWN′-256 (ref)	100021	105716	147664	170570	58737	27854
DAWN-α-1024 [LZL+25]	100021	103693	136020	167629	105594	131649

Table 6: Performance of polynomial arithmetic over R_2 on ARM Cortex-M4 (cycles).

Schemes	polymul_{64}	$\text{polymul}_{\frac{n}{4}}$	FastInversion	$\text{polymul}_{\frac{n}{2}}$	$\text{polymul}_{\frac{n}{4}\times\frac{n}{2}}$
DAWN′-128 (M4)	550	2260	19768	7440	4607
DAWN′-128 (ref)	3214	9920	63093	26655	26655
DAWN-α-512 [LZL+25]	3731	28489	134475	107957	107957
DAWN′-256 (M4)	550	7439	66903	23679	15057
DAWN′-256 (ref)	3214	32365	241087	120131	120131
DAWN-α-1024 [LZL+25]	3731	69807	540029	322100	322100

Polynomial arithmetic of R_2. Table 6 compares the performance of polynomial multiplication and inversion over R_2. We note that the multiplication routine in [LZL+25] is not constant-time, as its execution time depends on the structure of the input polynomials. When benchmarking this implementation using randomly sampled polynomial inputs, it is slower than both our constant-time Cortex-M4 implementation and the constant-time reference implementation.

For equal-degree polynomial multiplications across the evaluated sizes, our radix-16 implementation on the Cortex-M4 achieves speedups of 3.58×–5.84× over the conventional schoolbook implementation in the reference code. By combining radix-16 multiplication kernels for degrees $m = 2, 4, \ldots, n/4$ with tailored radix-16 routines for the remaining operations in fast inversion, we accelerate the fast inversion by 3.19× and 3.60× over the reference implementations of DAWN′-128 and DAWN′-256, respectively. Compared with the non-constant-time implementation of [LZL+25], the corresponding speedups are 6.80× and 8.07×, respectively.

In 2019, Bernstein and Yang [BY19] introduced an efficient constant-time algorithm for polynomial gcd computation and modular inversion. This algorithm was subsequently applied to NTRU on the ARM Cortex-M4 in [Li21]. A direct cycle-count comparison with our inversion routines is not entirely meaningful because the underlying quotient rings and polynomial degrees differ. Nevertheless, as useful reference points, polynomial inversion over $S_2 = \mathbb{F}_2[x]/(\Phi_n(x))$ requires 311,283 cycles for NTRU-HPS-2048-509 and 526,999 cycles for NTRU-HRSS-701 [Li21, Table 3.5]. These cycle counts are substantially higher than those of the fast inversion and base inversion routines in DAWN′ on the same platform, highlighting the low inversion overhead enabled by the arithmetic structure of DAWN′.

5.3 Performance Comparisons of the KEM Schemes

Combining the optimizations described above, as shown in Table 7, we obtain performance gains of 1.78–2.54× and 1.94–3.96× compared to the reference DAWN′ and DAWN [LZL+25], respectively. The results show that DAWN′-128 and DAWN′-256 achieve lower cycle counts on the ARM Cortex-M4 than the corresponding state-of-the-art implementations of the MLWE-based ML-KEM [HZZ+22] and the NTRU-family schemes NTRU-HPS, NTRU-HRSS, Streamlined NTRU Prime, and NTRU LPRime [AHY22, Che21]. Together with their compact key and ciphertext sizes, these results demonstrate that DAWN′ offers a favorable balance between communication cost and computational efficiency, making it a

promising KEM candidate for resource-constrained devices.

Table 7: Performance of KEM schemes on ARM Cortex-M4 (in kilo-cycles). Results on NUCLEO-L4R5ZI are averaged over 100 iterations. Results for other schemes are directly obtained from the original papers.

Scheme	Platform	Key generation	Encapsulation	Decapsulation
DAWN′-128 (M4)	NUCLEO-L4R5ZI	**303k** (1.00×)	**153k** (1.00×)	**288k** (1.00×)
DAWN′-128 (ref)	NUCLEO-L4R5ZI	594k (1.96×)	273k (1.78×)	570k (1.98×)
DAWN-α-512 [LZL+25]	NUCLEO-L4R5ZI	589k (1.94×)	406k (2.65×)	800k (2.78×)
ML-KEM-512 [HZZ+22]	NUCLEO-L4R5ZI	392k (1.29×)	390k (2.55×)	428k (1.49×)
NTRU-HPS$^{677}_{2048}$ [AHY22]	STM32F4-DISC	3912k (12.91×)	525k (3.43×)	718k (2.49×)
NTRU-HRSS-701 [AHY22]	STM32F4-DISC	3822k (12.61×)	361k (2.36×)	778k (2.70×)
NTRULPR-653 [AHY22]	STM32F4-DISC	669k (2.21×)	1131k (7.39×)	1231k (4.27×)
SNTRUP-653 [AHY22]	STM32F4-DISC	6623k (21.86×)	621k (4.06×)	527k (1.83×)
DAWN′-256 (M4)	NUCLEO-L4R5ZI	**630k** (1.00×)	**287k** (1.00×)	**557k** (1.00×)
DAWN′-256 (ref)	NUCLEO-L4R5ZI	1275k (2.02×)	612k (2.13×)	1413k (2.54×)
DAWN-α-1024 [LZL+25]	NUCLEO-L4R5ZI	1299k (2.06×)	861k (3.00×)	2204k (3.96×)
ML-KEM-1024 [HZZ+22]	NUCLEO-L4R5ZI	1018k (1.62×)	1031k (3.59×)	1094k (1.96×)
NTRULPR-1277 [Che21]	STM32F4-DISC	1433k (2.27×)	2023k (7.05×)	2212k (3.97×)
SNTRUP-1277 [Che21]	STM32F4-DISC	22853k (36.27×)	1320k (4.60×)	1066k (1.91×)

Acknowledgements

This work was supported in part by the National Key Research and Development Program of China (2023YFC3305503), the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (24-SIS-SMU-052), Scientific Research Innovation Capability Support Project for Young Faculty (SRICSPYF-BS2025137), Guangdong Provincial Key Laboratory of IRADS (2022B1212010006), Guangdong and Hong Kong Universities "1+1+1" Joint Research Collaboration Scheme (2025A0505000001).

References

[AAB+24] José Bacelar Almeida, Santiago Arranz-Olmos, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Cameron Low, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, and Pierre-Yves Strub. Formally Verifying Kyber - Episode V: Machine-Checked IND-CCA Security and Correctness of ML-KEM in EasyCrypt. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part II*, volume 14921 of *Lecture Notes in Computer Science*, pages 384–421. Springer, 2024.

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R, M} LWE schemes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):336–357, 2020.

[AHKS22] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 853–871. Springer, 2022.

[AHY22] Erdem Alkim, Vincent Hwang, and Bo-Yin Yang. Multi-Parameter Support with NTTs for NTRU and NTRU Prime on Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):349–371, 2022.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: Reducing Attack Surface at Low Cost. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, 2017.

[BKL+25] Manuel Barbosa, Matthias J. Kannwischer, Thing-Han Lim, Peter Schwabe, and Pierre-Yves Strub. Formally Verified Correctness Bounds for Lattice-Based Cryptography. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 156–169. ACM, 2025.

[Blu23] Bluetooth SIG. *Bluetooth Core Specification Version 5.4, Volume 3, Part F: Attribute Protocol (ATT).* Bluetooth Special Interest Group, 2023. Section 3.2.9, “Long Attribute Values”.

[BY19] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):340–398, 2019.

[CC21] Ming-Shing Chen and Tung Chou. Classic McEliece on the ARM Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):125–148, 2021.

[CCK21] Ming-Shing Chen, Tung Chou, and Markus Krausz. Optimizing BIKE for the Intel Haswell and ARM Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):97–124, 2021.

[CGKT22] Ming-Shing Chen, Tim Güneysu, Markus Krausz, and Jan Philipp Thoma. Carry-Less to BIKE Faster. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 833–852. Springer, 2022.

[Che21] Yun-Li Cheng. Number Theoretic Transform for Polynomial Multiplication in Lattice-based Cryptography on ARM Processors. Master's thesis, National Taiwan University, Jan 2021.

[CLWL26] Dongshu Cai, Yijian Liu, Jiabo Wang, and Xianhui Lu. Thorns in Polynomial Convolution: Correlation, Large Deviations, and Applications. Cryptology ePrint Archive, Paper 2026/1022, 2026.

[FKPY22] Pierre-Alain Fouque, Paul Kirchner, Thomas Pornin, and Yang Yu. BAT: small and fast KEM over NTRU lattices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):240–265, 2022.

[HAZ+24] Junhao Huang, Alexandre Adomnicai, Jipeng Zhang, Wangchen Dai, Yao Liu, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Revisiting Keccak and Dilithium Implementations on ARMv7-M. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):1–24, 2024.

[HRSS17] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. High-Speed Key Encapsulation from NTRU. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 232–252, Cham, 2017. Springer International Publishing.

[HZZ+22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Improved Plantard Arithmetic for Lattice-based Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):614–636, 2022.

[HZZ+24] Junhao Huang, Haosong Zhao, Jipeng Zhang, Wangchen Dai, Lu Zhou, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Yet Another Improvement of Plantard Arithmetic for Faster Kyber on Low-End 32-bit IoT Devices. *IEEE Trans. Inf. Forensics Secur.*, 19:3800–3813, 2024.

[KMRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM: CCA-secure module lattice-based key encapsulation on ARM. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):243–266, 2018.

[KRS19] Matthias J Kannwischer, Joost Rijneveld, and Peter Schwabe. Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates. In *International Conference on Applied Cryptography and Network Security*, pages 281–301. Springer, 2019.

[LGZ+26] Yijian Liu, Jiangxia Ge, Yu Zhang, Jiabo Wang, and Xianhui Lu. A Primer on Dependency in Polynomial Product: Identify, Exploit, and Trim. Cryptology ePrint Archive, Paper 2026/802, 2026.

[Li21] Ching-Lin Li. Implementation of Polynomial Modular Inversion in Lattice-based cryptography on ARM. Master's thesis, National Taiwan University, Sep 2021.

[LJKH25] MyeongHoon Lee, Jihoon Jang, Suhri Kim, and Seokhie Hong. Improved Frobenius FFT for code-based cryptography on Cortex-M4. *IEEE Internet Things J.*, 12(17):36318–36327, 2025.

[LZL+25] Yijian Liu, Yu Zhang, Xianhui Lu, Yao Cheng, and Yongjian Yin. DAWN: smaller and faster NTRU encryption via double encoding. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part III*, volume 16247 of *Lecture Notes in Computer Science*, pages 396–427. Springer, 2025.

[Nat24a] National Institute of Standards and Technology (NIST). FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard, August 2024. Federal Information Processing Standards Publication 203.

[Nat24b] National Institute of Standards and Technology (NIST). FIPS 204: Module-Lattice-Based Digital Signature Standard, August 2024. Federal Information Processing Standards Publication 204.

[PSH+25] Thales B. Paiva, Marcos A. Simplício Jr., Syed Mahbub Hafiz, Bahattin Yildiz, Eduardo Lopes Cominetti, and Henrique S. Ogawa. Tailorable codes for lattice-based kems with applications to compact ML-KEM instantiations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(3):139–163, 2025.

[SKLE25] Minjoo Sim, Hyunjun Kim, Minwoo Lee, and Siwoo Eum. High-Speed 16-Radix Polynomial Multiplication on ARM Cortex-M4 with Recursive Karatsuba Layers. In *International Conference on Information Security and Cryptology*, pages 385–399. Springer, 2025.